From Problem Frames to HJJ (and its known unknowns)

Cliff B. Jones

School of Computing Science, Newcastle University, UK

Draft Dated: July 5, 2009

Abstract. This paper traces the evolving ideas of an approach to "open systems" that interact with the physical world. Such systems nowadays almost always include computers. The design in the simplest cases has a control computer connected to sensors that receive information about the physical world and to actuators that can cause some aspects of that world to change. Jackson's "Problem Frame Approach" is to think about the computer system with respect to the requirement of the desired overall system behaviour; the "Hayes/Jackson/Jones" (HJJ) approach introduces sufficient formalism to support the derivation of the specification of the computer control system.

1 Introduction

This paper is an update of on-going research that Ian Hayes and I have the pleasure of undertaking with Michael Jackson. After the initial letter of each of our family names, the research will be referred to as the "HJJ" approach (citations below). The HJJ ideas owe a huge debt to Michael's "Problem Frame Approach" (PFA) [Jac03] (and PFDs [Jac00] in general). This section sets out the background; Section 2 outlines those parts of the HJJ approach with which we are moderatly satisfied; the problem areas are sketched in Section 3; and our current ideas on filling the gaps are mentioned in Section 4.

The three authors of [HJJ03,JHJ07] share the –not uncommon– view that the process of software development is imperfect. That having been said, it is also our view that the theory for development of "closed" systems is known. By a closed system, we mean one that exists in some neat domain such as number theory or matrix algebra. Here one can write a brief and complete specification of what it means, say, to print all of the primes less than a million or to invert a matrix. Furthermore, ideas like data reification[1] and operation decomposition [Jon90] can be employed to show that steps of design lead from such a specification to code that satisfies the specification. (If this were to be a complete

[1] I owe thanks to Michael for pointing out that the transition from a clean mathematical abstraction such as sets to a machine implementation based (necessarily) on a mess of pointers should not be considered to be "refinement". Sadly, few others have seen the force of the observation and adopted my use of "reify" as a more appropriate verb.

account, we should add comments about compositionality, assumptions and acknowledge that real development projects undergo requirements changes during development.)

The situation is rather different with "open systems" where it is the physical world that both sets the context for the system and –in some sense– provides the touchstone of acceptability of the system. A classic example that we have (over) used is that of a "sluice gate".[2] The overall requirement might be that the ratio of open to closed time of the physical gate approximates to some required ratio. A more challenging example might be an advisory system for air traffic controllers whose ultimate responsibility is to achieve minimum vertical and horizontal separation of physical aircraft.

Such open systems are the meat of Michael's Problem Frame Approach (PFA) [Jac03,Jac00]. The insistence in PFA in grounding the requirements of a system in the physical world is central to HJJ.

At most of the points on the complexity spectrum from sluice gates to air traffic control systems, the physical phenomena are likely to vary continuously. Various notations have been used for such descriptions. Perhaps the best known is the "Duration Calculus" [CHR91,CH04]. HJJ has deployed the notation proposed by Ian Hayes and Brendan Mahony [MH91,MH92] (which references also begin to use the ideas in the next paragraph). Key to the usefulness of such notations for continuous variables is the ability to use a form of integration that makes it possible to express things like the amount of time, perhaps over a stated interval, that a value has certain properties.

My own contribution to HJJ derives from earlier work on the development of concurrent programs: [Jon81,Jon83a,Jon83b] show that it is possible to use rely and guarantee conditions to achieve compositional development even in the presence of "interference". The specific rules offered in these early papers are not the point. More recently, what might be called "rely/guarantee thinking" [Jon96,CJ00,CJ07,JP08] generalises the approach to the rather obvious position that if one wants to reason compositionally about interfering systems, one must record something about the interference. We have used rely/guarantee thinking in HJJ to record *assumptions* about those physical components that are part of the overall system but are not included in the software system that is to be designed.

That's enough background, it is time to outline the HJJ approach in as far as we are confident about its features.

[2] For readers unfamiliar with [Jac00] (or the HJJ papers themselves) a few words about this example should suffice. Consider an irrigation channel into which the flow of water is controlled by a sluice gate. This gate has sensors at the extreme positions. There is also a bi-directional motor that will raise or lower the gate. As well as being connected to the gate, the motor has actuators which can set its direction and turn it on or off. The sensors and actuators are connected to a control computer which can thus (indirectly) cause the gate to move safely between extreme positions.

2 What we (think we) know

For open systems, a major issue is how a specification is obtained. Think of a requirement to maintain the temperature in some space within certain bounds. Software alone can have no affect on the temperature of anything but this does not mean that one should immediately begin writing the specification of a control system as an isolated component. It is basic to PFA to ground the system in its overall physical environment. In HJJ as well, we begin by writing the requirement of the overall system at the level of the physical phenomena: that is, the temperature of the room. If one forgets what one knows about a particular geographic location, it is obvious that it is necessary to make *assumptions* about how fast the contextual (ambient) temperature can change. So, from the outset, one is faced with recording some assumptions. To arrive at the specification of what Michael calls "the silicon package" (i.e. the control system), it is necessary to record more assumptions.

It is the essence of the HJJ approach to "derive" the specification of the silicon package from that of the overall system. Two of the contributors to HJJ are unreformed formalists so they strive for notations with which all of the specifications and assumptions can be made precise (and tractable) enough that proofs can be written. The extra assumptions that are required to make such a derivation possible are about both the way in which the software can obtain values of things in the physical world and how sending signals might cause other (physical) components to affect the real world. In fact, the assumptions often twine together the outputs and inputs. In the case of a temperature control system, one needs assumptions about sensors that convert the temperature of the physical environment into signals that can be read by the computer on which the software is executed and assumptions about the effect (on the temperature of the room) of sending signals via actuators to heating and/or cooling devices.

The HJJ approach then is to begin with a requirement grounded in the physical world; to record assumptions about how the world can change; to record further assumptions about physical components of the system; to "derive" the specification of the control system (silicon package); and to check that an implementation of the silicon package that fulfils its specification, in an environment that satisfies the assumptions, will indeed keep the overall system within the bounds required in the physical context. (The reader might well wonder what happens if these assumptions are undermined by failure of some physical component; this question is addressed below.)

In nearly all of the systems we have tackled with the HJJ approach, this reasoning ends up being about continuously varying quantities. For example, in the sluice gate problem in [JHJ07], the physical phenomena are indexed by continuous time. Many providers of formal methods tend to what might be termed "premature discretisation"; because their method revolves around discrete operations (or events), they assume some arbitrary polling frequency to reduce the understanding of the continuous phenomena to discrete behaviour. For people who will extol for hours the advantages of formal reasoning this is interesting behaviour. It should be clear that questions like the polling frequency need to be

reasoned about in terms of assumed rates of change etc. It is almost inevitable that one ends up at a discrete controller but the derivation of its specification should be part of the (formal) development process.

It is important to note that we are here "preserving the design history" in the same sense that standard formal stepwise refinement leaves behind a record of design steps and their justification.

It is *crucial* to understand that the HJJ process does *not* require that the designer build a complete model of the physical components of the system. Again, in the sluice gate example, assumptions are made about the position sensors returning indicators that the gate has achieved a required physical position within a certain period of time from sending a polarity setting signal followed by a start motor signal. This circumvents the need to build a model of motors, gears etc.

So, to return to the issue at the beginning of this section (how to obtain a specification of a control system that is intended to interact with the physical world), the basic HJJ approach can be summarised as:

- determine the requirements of the overall system with respect to the physical phenomena
- record assumptions about the way those phenomena can be affected by things beyond the system
- record assumptions about the physical components of the system (linking the external world to the silicon package via sensors and actuators)
- "derive" a specification of the silicon package that will achieve the overall requirement if the other system components behave according to their assumptions

Of course, the overall requirement *and the assumptions about other components* must be agreed with the customer. But it is a key advantage of the HJJ approach that it leaves a record of assumptions for subsequent deployments. In contrast to any approach that tries to build a model of the environment, the HJJ message might be viewed as "how little can we say (about the environment)?".

Interestingly, we have found many examples where rely conditions are used within a development to record assumptions that it is necessary to make for safe use of the physical components of the system. For example, in the sluice gate, one must not reverse the polarity of the gears connecting the motor to the gate whilst the motor is running.

Most importantly, the HJJ approach councils against a premature jump into the specification of the silicon package. Such a jump nearly always results in a confusion about assumptions and requirements. Examples of formal specifications in the literature where this is the case are too numerous to list.

This much of the HJJ approach was fairly well established in [HJJ03] and is most clearly set out [JHJ07]. Both papers use the sluice gate problem which is, among other reasons, interesting because it also has to address fault-tolerant behaviour. At one level, addressing degraded modes of behaviour requires no more than recording weaker assumptions about the environment but as is conceded in the next section, there is a difficulty in knowing the best way to combine the specifications of normal and error recovery modes.

Before moving on to the issues that still need to be resolved, it is worth *sketching* what the HJJ approach might achieve for one or two aspects of a complex system like an advisory system for air traffic controllers. Suppose that a base air traffic control system is in place: RADAR, displays of aircraft position, etc. are all established; assume further that one then wished to specify a new system that helped Air Traffic Controllers (ATCs) by warning of future approaches within specified distances and –furthermore– to carry out thought experiments like "what are the effects of sending an aircraft on a particular vertical angle and acceleration". Given the large system that already exists, the obvious temptation is to use the expertise of people who know the existing system and the physical behaviour of aircraft and to simply specify the new component. The HJJ approach would advise against this; or rather would suggest that there are advantages in a more circuitous route to the specification of the new component (as a separate silicon package).

What the ATCs are required to achieve is physical separation of aircraft.[3] This is a sobering place to start: an ATC does not, in fact, move aircraft. Thus one sees at once that a fairly simple overall requirement is actually made challenging (to meet) because of the assumptions that have to be made. Interestingly, the assumptions in this case (as in many complex systems) involve some on human players: it is not an actuator that connects the silicon package to the position of the aircraft but rather the ATC sending a voice message to a pilot who might or might not react as intended. Such assumptions must be recorded.

Furthermore, there are many assumptions to be made about the physical ability of an aircraft to achieve particular changes of position. Factors that influence this include type, load and air pressure. Focusing on this last item, the approach of jumping straight to a specification of the new sub-system might well make inadvertent assumptions about the potential rate of change in air pressure at a particular geographic location. This in turn might result in reading air pressure at some fixed intervals.

If, instead, one determines the potential effect of each of the factors on maneuverability of aircraft, one can calculate what might have to be considered to achieve certain tolerances; one can *record* assumptions about changes in air pressure; and prove that a certain currency of information is adequate. Here again, the recorded assumptions could be invaluable if the same software were to be deployed in another geographic location.

[3] Strictly, there are also progress conditions: the easiest way to achieve separation is to have few –or no– aircraft in the sky.

3 What we know that we don't know

Dubium Sapientiae initium[4]

How far should one push out the boundaries of a system?

We have argued that the HJJ approach (just as PFA) should start by pushing out the boundaries of "the system" to be considered — but how far? This is certainly a valid question about HJJ but it is in some sense unanswerable. To continue with the imaginary air traffic scenario, if one were to regard the system as that of the planet, one might indeed decide that safe aircraft separation was best achieved by flying less planes thus obviating the need for ways to inform ATCs of ways to handle more flights. It is however an answer that is unlikely to impress National Air Traffic Services (NATS is the UK body responsible for air traffic in the UK airspace).

One can see that the approach of recording assumptions comes into play wherever one places the boundary by returning to the example that our papers have analysed in more detail:

- Setting the boundary at moving the sluice gate prompts the need for *assumptions* about the relationship between request signals (presumably causing actuators and motors to perform) and return signals from sensors with further assumptions about the relationship between the sensor values and phenomena in the physical world.
- The sluice gate was proposed by Michael as an object of study in [Jac00] as a way of controlling the irrigation of fields — were one to view the *requirements* at that level, *assumptions* would be needed about the presence of water at the gate.
- Presumably the real reason for releasing the irrigation stream is to achieve a certain moisture level in the soil — one could only achieve a satisfiable specification at this boundary by making assumptions about the weather.
- One can go yet further and discuss things like farm subsidies if the requirement is about farm profits.

With typical pragmatism, Michael offers the only insight possible as to where one should stop this broadening: if the customer is the farmer –and he says "the job is to achieve a certain gate open/close ratio"– then that bounds the concern of the developer. It is however true that one can sometimes get a clearer view by considering a slight extension that grounds the purpose without undermining the legitimate requests of the paying customer. For example, knowing that the gate is for irrigation might prompt a wise designer to make the actual ratio of open vs. closed times a rather easy parameter to change in the created system. There are also overriding ethical considerations that would, hopefully, prevent most designers building systems whose function was manifestly immoral.

[4] As a classics scholar like Michael can tell you, Descartes was somewhat ahead of Donald Rumsfeld (or even Thoreau "To know that we know what we know, and that we do not know what we do not know, that is true knowledge") in pointing out that "Doubt is the origin of wisdom".

Linking fault tolerant specifications to the idealised behaviour

A technical challenge that has occupied the three authors for some time is handling the sort of exceptional behaviour that is often seen in fault tolerant systems. For any system there is in our opinion a strong case for recording first its idealised behaviour. The term "idealised" is meant to cover the obvious functionality which is only likely to be achievable under optimistic assumptions such as all of the physical components working without failure. The issue that in a sense represents the difference between the two HJJ papers ([HJJ03] and [JHJ07]) is how one tackles adjoining the specification of the exceptional behaviour to that of the idealised system. Unfortunately, we do not feel that the ideas in the more recent paper are a total solution — see Section 4.

To return to the sluice gate example, suppose the assumption that turning the motor on should achieve a physical change in the gate position appears to be false: after a sensible period, the sensor at the extreme to which the gate was to be propelled has not changed state. In order to avoid burning out the motor by pushing the gate against a physical limit, it is not difficult to add a requirement that the motor should never be run for more than, say, a minute. If one is lucky enough to have a free hand in ordering further sensors, one might track extra phenomena; even if not, a developer might well suggest a warning signal be added to record when normal operation ceases.

This train of thought is what gives plausibility to the statement that handling exceptions just needs weaker assumptions and different (in some sense also weaker) guarantees. But there is a difficult question about handling the handover between normal and exceptional behaviour. If one wants a single specification of the entire system, it is very easy to be inadvertently prescriptive about exactly when the deviation from normal behaviour must be sensed. This key problem is addressed in Section 4 as describing "phase changes".

Michael's relatively small challenge problem offers another sort of exception that has taxed the basic HJJ approach. The other side of the coin on delayed sensor indication is a premature signal. Indeed, what if sensors indicating that the gate is in two different positions are on "together". Clearly, something is amiss. It is tempting to say that a safe system must detect such abnormalities (and again turn on an alarm). But here again, stating the precise timing can result in unrealisable specifications. For example, if the eventual program is to poll two sensors, it cannot possibly detect that the two sensors are on at precisely the same time because the polling events will be separated by an –admittedly very small– interval. In fact, two such positive sequential sensor readings (separated perhaps by a thread interruption) do not constitute proof that the sensors were on unacceptably close together in time. Of course, in the example of the sluice gate, with presumably a dedicated processor, these issues could probably be ignored. But they would be ignored at risk to life and limb if one were designing an in-flight monitor or a reactor protection system.

There should clearly be a general way to record and reason about the kind of transient issues that have been illustrated here with short-circuit or flickering sensors. In [JHJ07] we made some progress on this issue with a form of "Deon-

tic implication" stating that (a) if errors persist for a long period of time, the system must shut down; and (b) that the system is allowed to shut down only if there was at least a (short) period of time when evidence for erroneous behaviour could be observed. In that paper we also looked at other ways of making system descriptions more robust by minimising assumptions and considering the gathering of evidence that a system is getting closer to violating its assumptions.

These topics are looked at again in Section 4 but with only indications of how we hope to achieve a perspicuous combination of idealised and exceptional behaviour.

Stochastic

Many of the fault-tolerance issues are in fact likely to be expressed in terms of probabilistic (e.g. mean time to failure) terms. For example, the overall requirement of a highly reliable sluice gate might state how often the customer is prepared to accept a failure; an implementation might need "triple modular redundancy" on the sensors to achieve the requirement. Clearly, this is more likely to be a significant issue towards the air traffic control end of the complexity spectrum. There, it is likely to be the case that one has to make stochastic assumptions about the behaviour of human players in a system. Interestingly, the role of humans also becomes important in thinking about the dependability of systems that use encryption for security (e.g. the trade-off between longer passwords being more secure and the ability of humans to remember them without recording them in ways that increase overall system vulnerability).

The only observation that is made in this paper is that it looks plausible to extend rely/guarantee thinking to stochastic statements — but there is no corresponding item in Section 4 that takes this point forward.

Making HJJ into a method

The reader might have noted that HJJ has been referred to as an "approach". There have been erudite discussions between Michael and my colleague at Newcastle, Manuel Mazzara, as to what constitutes a "method". (In fact, these discussions gave me the Latin quotation from Descartes.)

What we can say so far is that an outline of an HJJ method might include:

1. choose the system's perimeter (wider than the pure control system)
2. record (and agree with the customer) the requirements at this level
3. record (and agree with the customer) assumptions about other system components necessary for this behaviour to be realisable
4. iterate steps 2–3 with weaker assumptions to cover exceptional behaviour
5. handle the "composition issue"[5]

Here again, this point is not developed in the current paper. The reader is referred to [Maz09] for a discussion of "Layered Fault-Tolerant Systems". It is also worth remembering that Michael had the requirement that a method in the sense of [Jac75] or [Jac83] should in some reasonable sense be "normative".

[5] See Section 4.

Hasn't this all been done before?

When undertaking research, I prefer to have a go at a problem without reading other "possible solutions" — but there comes a point at which it is prudent to check if someone else has solved a problem that, in this case, has held up the HJJ authors for a number of years. We would in fact be delighted to find such a preemption and move on to other topics.

To make clear why we feel this is not a solved problem, it is perhaps worth reiterating some of our desiderata.[6] I have long held the view that it is possible to view the (possibly erroneous) behaviour of external components as a form of "interference". This dates back to some consulting work I did on an "Inherently Safe Automatic Trip" [SW89]: briefly, ISAT was a clever nuclear reactor protection trip system that had many internal checks (and caused reactor shut-down if these failed) but there was no proof of resilience against a *stated* set of exceptions. We showed that the assumptions could be captured by rely conditions. One thing that inhibited wider publication at that time was the fact that the assumptions tended to be at a different level of abstraction than the system functionality.

It is a distinctive feature of the HJJ approach that it is not intended that a model is built of the external system components; the aim is to make *minimal* assumptions.

It is also important to note that the HJJ approach does not construct a state in which both physical phenomena and their approximation are stored together. This feels like the same sort of confusion that dogged programming language descriptions that used "grand state" operational semantic descriptions. The HJJ approach records assumptions about the relationship between things beyond the machine and their internal representations (approximations). In so doing, it makes immediately clear that the machine cannot (directly) affect the values in the physical world. It is interesting to link this thought with the writings of another wise author: [Ken78] makes the clear distinction between "Data and Reality".

4 Where next

The frank list of open issues in Section 3 might appear off-putting. In this section, we indicate where we are looking for progress. Hayes and Jones want a way of recording the overall system specification. Furthermore, this should not consist of a massive conjunction of implications.[7] In contrast, Michael Jackson is, in cases of "normal design", prepared to leave some of the overall system properties to known engineering. (More recently, Michael has also drawn inspiration from Polanyi's writings about "operational principles".) This is not to say

[6] For some more detailed comparisons with other approaches, see [JHJ07, Section 4.1].

[7] Inspiration here might be taken from the **err** clauses in VDM [Jon90]. In fact, this is rather close to the **otherwise** operator with which we experimented in [HJJ03].

that Michael would reject an overall system description but he is perhaps less concerned about it than his two co-authors.

In any case, it is clear that a key to perspicuous ("layered") specifications of fault-tolerant systems is the ability to handle "phase change" conveniently. The notion of "phases" has other applications. It is for instance possible to consider the important "initialisation concern" via phases; it is also possible to consider the broader lifetime of a system including commissioning, revision and decommissioning from the point of view of phases.

These considerations have led us to look at the "Time Bands" concept described in [BH09]. In outline, like most good ideas, what underlies time bands is rather simple: it is often easier to understand (and/or describe) a system at different granularities of time. What is most easily viewed as an atomic "event" at a coarse granularity might be broken down at a finer granularity. Central to the time band model of [BH09] is the idea that there is a "precision" associated with each time band: when one says –at say the hour band– that a meeting starts at 11.00, there is no expectation this has to happen to the second. One intriguing match between time band concerns and the handling of exceptional behaviour is the way in which exceptions are often distinguished by crossing time bands.

The model developed in [BH09] is to my eyes more complicated than is needed to resolve the open issues in HJJ.[8] In fact, just the inclusion of a notion like (time) precision might resolve the phase change issue.

Anther intriguing avenue is the teleo-reactive notation of Nilson [Nil94] as developed by Keith Clark. Ian Hayes [Hay08] has observed that the way in which outer guards are evaluated as the state evolves models closely the way in which exceptions cause interruption of "durative" actions. (Ian has also observed that in some sense this just shifts the semantic problem; but at least the difficulty is concentrated in one place.)

Acknowledgements

Anyone who has cooperated with Michael Jackson knows how stimulating it can be: the untiring series of challenges and the joy in discussing them (always with supreme politeness) makes him one of the most productive collaborators I know.

Since this paper is written (unusually for me) in the first person singular, I should also like to record my thanks to the other person in HJJ: Ian Hayes is ever patient and thorough — also a joy as a collaborator.

My research is currently funded by the EU "Deploy" project, the (UK) EPSRC "TrAmS" project and the ARC project (that brings together Ian Hayes, Keith Clark, Alan Burns and myself) "Time Bands for Teleo-Reactive Programs".

[8] We have discussed questions like "timeless time bands" and using the "fuzziness" notion on all measures not just time.

References

[BH09] Alan Burns and Ian Hayes. A timeband framework for modelling real-time systems. *submitted to Real-Time Systems*, 2009.

[CH04] Zhou Chaochen and Michael R. Hansen. *Duration Calculus*. Springer, 2004.

[CHR91] Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–271, December 1991.

[CJ00] Pierre Collette and Cliff B. Jones. Enhancing the tractability of rely/guarantee specifications in the development of interfering operations. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language and Interaction*, chapter 10, pages 277–307. MIT Press, 2000.

[CJ07] J. W. Coleman and C. B. Jones. A structural proof of the soundness of rely/guarantee rules. *Journal of Logic and Computation*, 17(4):807–841, 2007.

[Hay08] Ian J. Hayes. Towards reasoning about teleo-reactive programs for robust real-time systems. In *SERENE '08: Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems*, pages 87–94, November 2008.

[HJJ03] Ian Hayes, Michael Jackson, and Cliff Jones. Determining the specification of a control system from that of its environment. In Keijiro Araki, Stefani Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *Lecture Notes in Computer Science*, pages 154–169. Springer Verlag, 2003.

[Jac75] Michael Jackson. *Principles of Program Design*. Academic Press, 1975.

[Jac83] Michael Jackson. *System Design*. Prentice-Hall International, 1983.

[Jac00] Michael Jackson. *Problem Frames: Analyzing and structuring software development problems*. Addison-Wesley, 2000.

[Jac03] Michael Jackson. Aspects of system description. In Annabelle McIver and Carroll Morgan, editors, *Programming Methodology*, pages 137–160. Springer Verlag, 2003.

[JHJ07] Cliff B. Jones, Ian J. Hayes, and Michael A. Jackson. Deriving specifications for systems that are connected to the physical world. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays*, volume 4700 of *Lecture Notes in Computer Science*, pages 364–390. Springer Verlag, 2007.

[Jon81] C. B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, June 1981. Printed as: Programming Research Group, Technical Monograph 25.

[Jon83a] C. B. Jones. Specification and design of (parallel) programs. In *Proceedings of IFIP'83*, pages 321–332. North-Holland, 1983.

[Jon83b] C. B. Jones. Tentative steps toward a development method for interfering programs. *Transactions on Programming Languages and System*, 5(4):596–619, 1983.

[Jon90] C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, second edition, 1990.

[Jon96] C. B. Jones. Accommodating interference in the formal design of concurrent object-based programs. *Formal Methods in System Design*, 8(2):105–122, March 1996.

[JP08] Cliff B. Jones and Ken G. Pierce. Splitting atoms with rely/guarantee conditions coupled with data reification. In *ABZ2008*, volume LNCS 5238, pages 360–377, 2008.

[Ken78] William Kent. *Data and Reality*. North Holland, 1978.

[Maz09] Manuel Mazzara. Deriving specifications of dependable systems: toward a method. In *Proceedings of the 12th European Workshop on Dependable Computing (EWDC 2009)*, 2009.

[MH91] B. Mahony and I. Hayes. Using continuous real functions to model timed histories. In P. Bailes, editor, *Engineering Safe Software*, pages 257–270. Australian Computer Society, 1991.

[MH92] B. P. Mahony and I. J. Hayes. A case-study in timed refinement: A mine pump. *IEEE Trans. on Software Engineering*, 18(9):817–826, 1992.

[Nil94] N. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[SW89] I. C. Smith and D. N. Wall. Programmable electronic systems for reactor safety. *Atom*, (395), 1989.